Introducción a Ruby

Scopes

Overview

- Scope de las variables
- Scope de Constants
- Cómo funciona el scope con los bloques

Scope

- Los métodos y clases inician un nuevo scope para las variables.
  - No pueden ver las variables del código que las invoca.
- Se puede utilizar el método local_variables para conocer cuales son las variables que están visibles en el scope actual.

Scope de Variables

```ruby
v1 = "outside"

class MyClass
  def my_method
    # p v1 EXCEPTION THROWN - no such variable exists
    v1 = "inside"
    p v1
    p local_variables
  end
end

p v1 # => outside
obj = MyClass.new
obj.my_method # => inside
              # => [:v1]
p local_variables # => [:v1, :obj]
p self # => main
```

Scope de Constantes

- Una **constante** es definida por cualquier referencia que inicie con una letra **mayúscula**, incluyendo las **clases** y los **módulos**.
- Las reglas de scope de las constantes son **diferentes** a las reglas de scope de las variables.
- Los scope internos **pueden ver** las constantes definidas en el scope externo e incluso puede sobreescribir el valor de las **constantes externas**.
    - El valor permanece sin cambios en el scope externo.

Scope Constantes

```ruby
module Test
  PI = 3.14
  class Test2
    def what_is_pi
      puts PI
    end
  end
end
Test::Test2.new.what_is_pi
# => 3.14
```

```ruby
module MyModule
  MyConstant = 'Constante externa'
  class MyClass
    puts MyConstant
    # => Constante externa
    MyConstant = 'Constante Interna'
    puts MyConstant
    # => Constante interna
  end
  puts MyConstant # => Constante externa
end
```

Scope de bloques

- Los bloques heredan el scope externo.
- Un bloque es un closure lo que quiere decir que recuerda el contexto en el cual fue definido y utiliza ese contexto cada vez que es llamado...

Scope de Bloques

```ruby
class BankAccount
  attr_accessor :id, :amount
  def initialize(id, amount)
    @id = id
    @amount = amount
  end
end
```

```ruby
total_sum = 0
accts.each do |eachAcct|
  total_sum += eachAcct.amount
end

puts total_sum # => 200
```

```ruby
acct1 = BankAccount.new(123, 200)
acct2 = BankAccount.new(321, 100)
acct3 = BankAccount.new(421, -100)
accts = [acct1, acct2, acct3]
```

Scope Local de un Bloque

- A pesar de que los bloques **compartan** el scope externo, una variable creada dentro de un bloque es **solo visible para el bloque**.
- Los **parámetros** de un bloque **son siempre locales al bloque** incluso si estos parámetros tienen **el mismo nombre** que algunas variables en el scope externo.
- Se pueden **declarar explícitamente** variables locales-al-bloque después de colocar un **;** en la lista de parámetros.

Scope local de un bloque

```ruby
arr = [5, 4, 1]
cur_number = 10
arr.each do |cur_number|
  some_var = 10 # No esta disponible fuera del bloque
  print cur_number.to_s + " " # => 5 4 1
end

puts cur_number # => 10
```

```ruby
adjustment = 5
arr.each do |cur_number;adjustment|
  adjustment = 10
  print "#{cur_number + adjustment} " # => 15 14 11
end
puts
puts adjustment # => 5 (No está afectado por el bloque)
```

Entonces...

- Los métodos y las clases inician un nuevo scope.
- Las constantes trascienden a scopes internos.
- Los bloques heredan el scope del código que los invoca.
  - Puede sobre-escribir variables del scope externo.